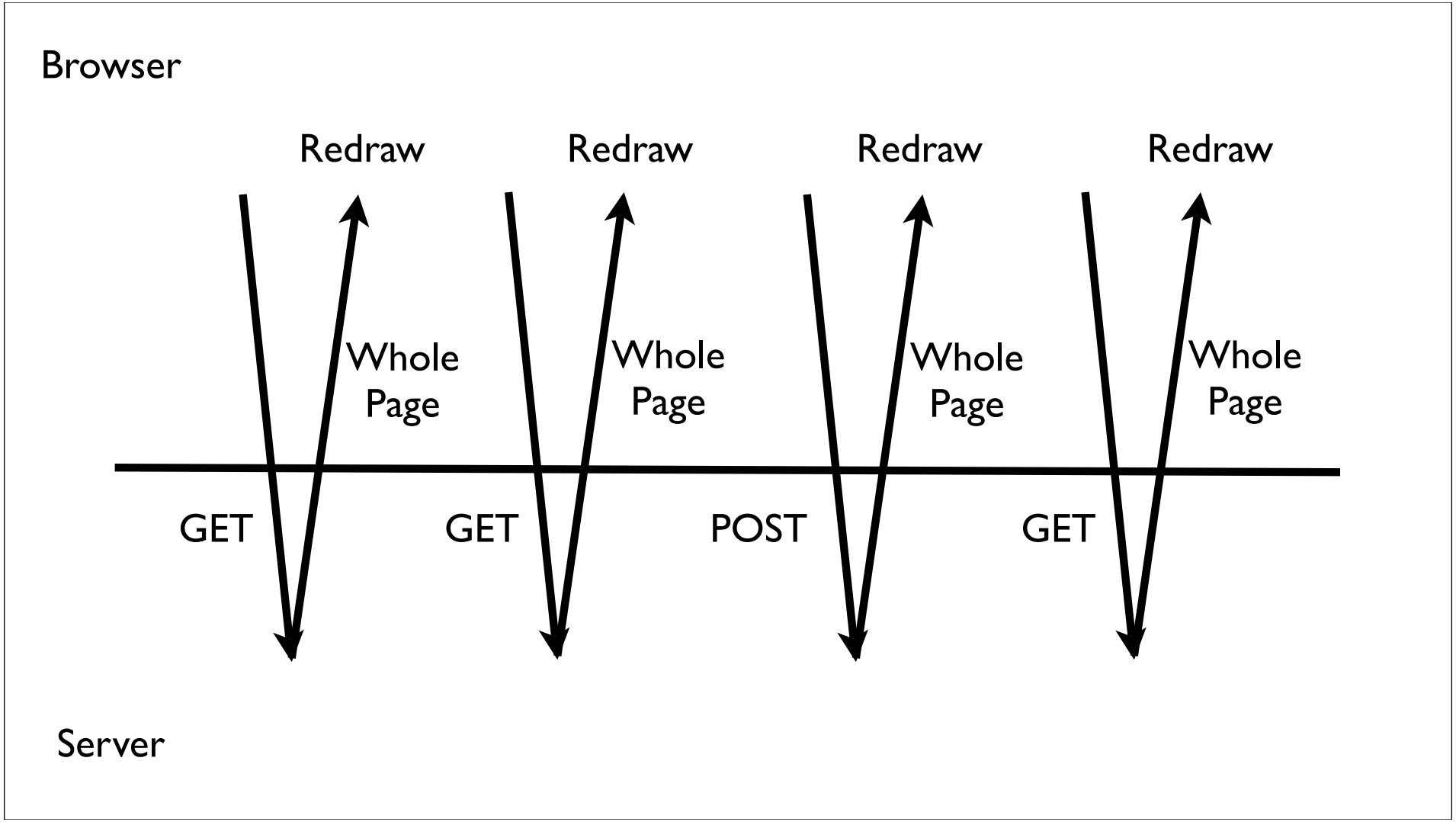
Ajax

SI539 - Charles Severance

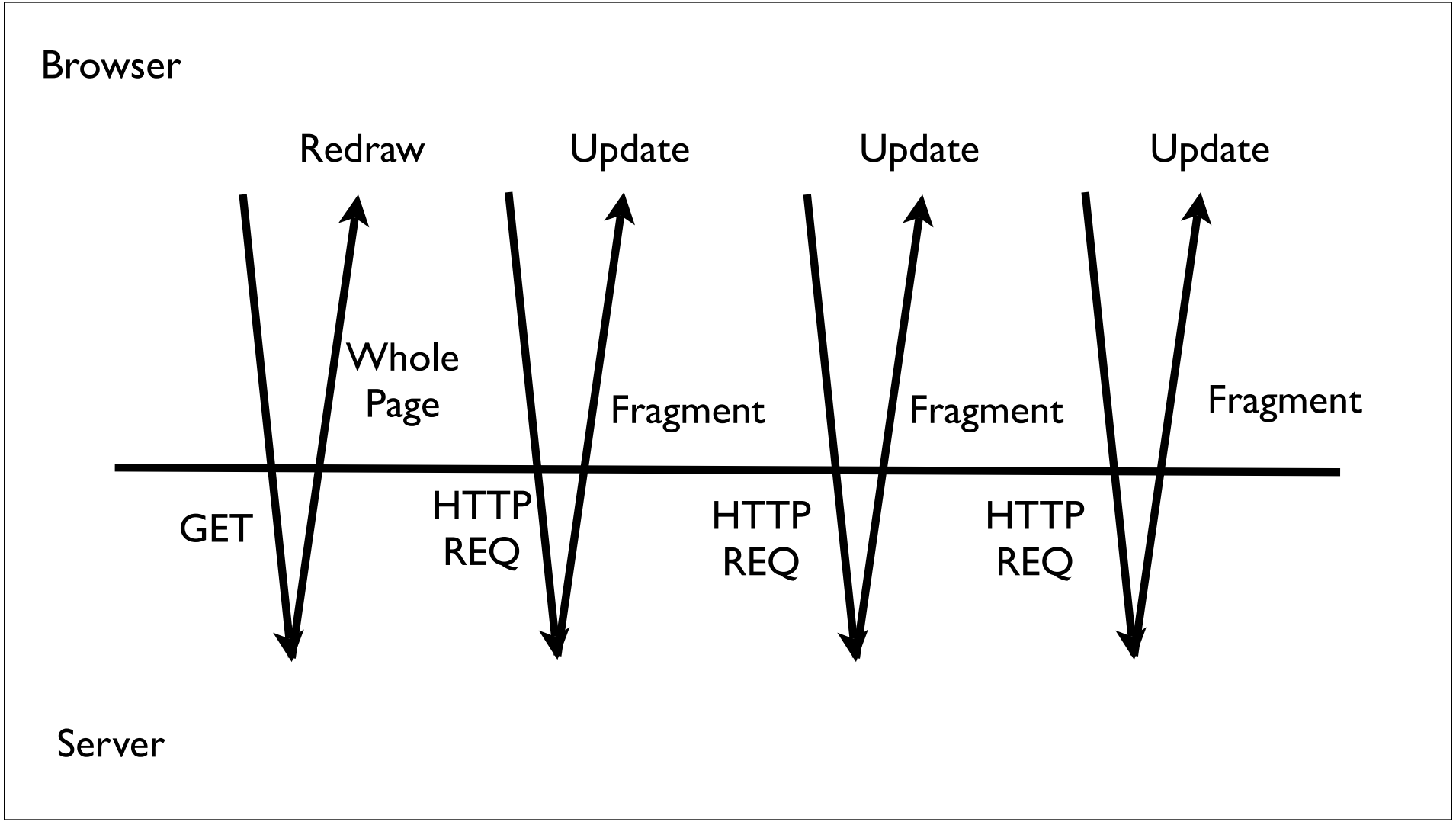
In The Good Old Days

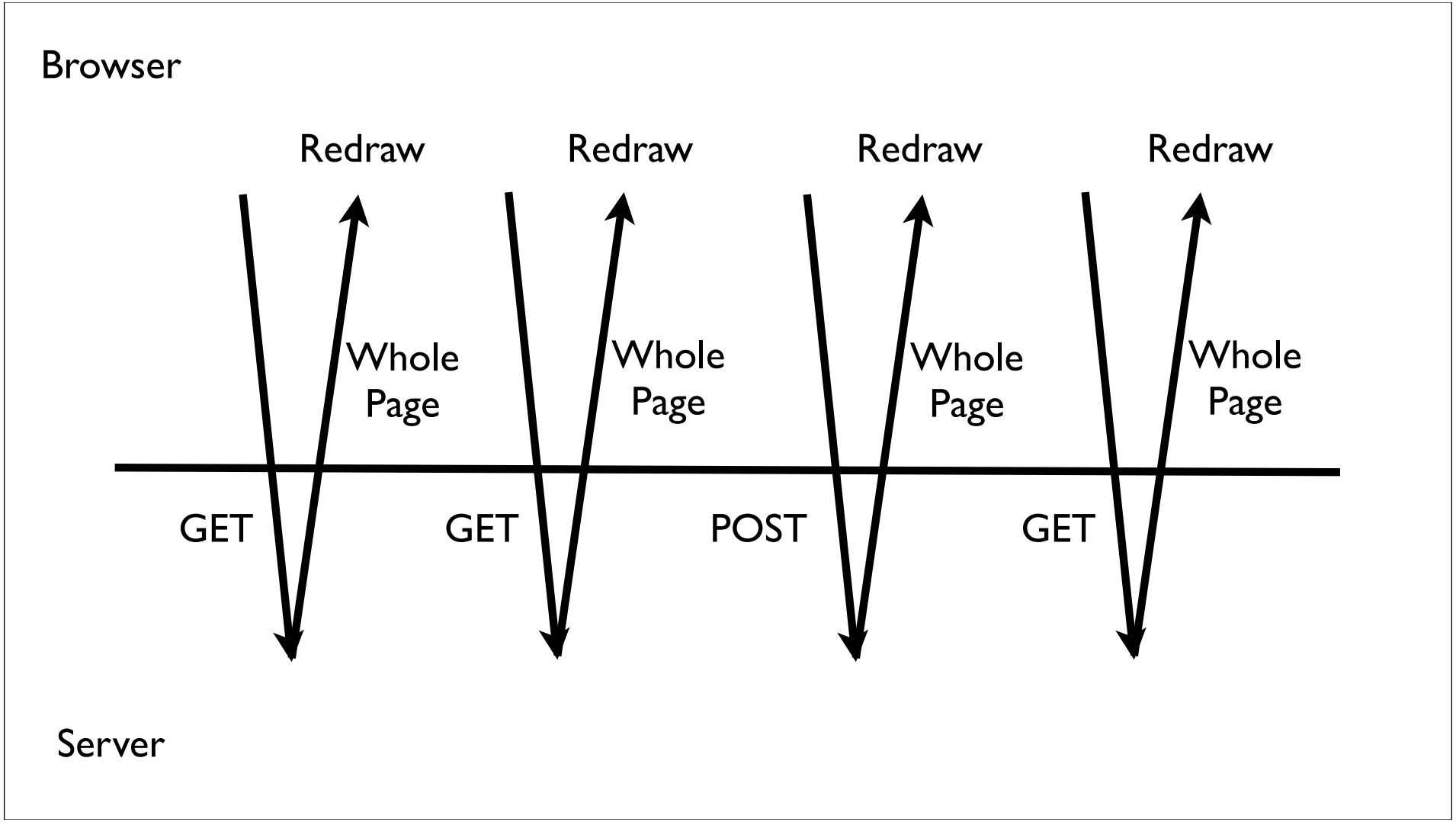
- A user would take some action like a click on a link or button
- The Browser would make a TCP/IP conection to the web server
- The browser would send a POST or GET request
- The Server would send back a page to display to the user
- Repeat...


Browser
Redraw
Redraw
Redraw
Redraw
Whole Page
Whole Page
Whole Page
Whole Page
GET
GET
POST
GET
Server

XMLHttpRequest

- In 2000, Microsoft wanted to move some of the processing of web pages from the web server to the web browser
- The idea was instead of sending whole pages of HTML to the browser, instead send out the data to be displayed in XML and then produce presentation in Javascript in the browser
- Originally a Microsoft innovation - other browsers soon adopted the idea and it became a defacto standard with a little variation between browsers :)
- It soon became clear that this could send *anything* - not just XML back and forth between a browser and client


Browser
Redraw
Update
Update
Update
Whole
Page
Fragment
Fragment
Fragment
GET
HTTP
REQ
HTTP
REQ
HTTP
REQ
Server


Browser
Redraw
Redraw
Redraw
Redraw
Whole Page
Whole Page
Whole Page
Whole Page
GET
GET
POST
GET
Server

Ajax Arms Race

- The race was on to build better and better web sites that began to replace things like frames and full-screen updates with XHtmlRequest operations and very selective screen updates.
- With clever Javascript programmers - the impossible became possible - drag and drop, automatic field completion - automatic data saving. It made the web operate much more like the desktop.
- Applications like GMail and Google Maps - feel very un-web.

Ajax versus Request/Response

- Standard Request/Response
  - Each click presents a whole new screen
- Ajax
  - Each click sends data and receives results in the background.
  - The browser typically gets back a fragment of HTML which is used to update a portion of the screen using the browser document model

```
<p>
<a href="#"
  onclick="document.getElementById('stuff').innerHTML = 'BACK';">BACK</a>
<a href="#"
  onclick="document.getElementById('stuff').innerHTML = 'FORTH';">FORTH</a>
</p>
<p>
Hello <b><span id="stuff">Stuff</span></b> there.
</p>
```

Updating the Browser Document

This is why you can only have one id per document.

Lots of Flexibility

- When you combine the ability to rewrite the Browser document model with the ability to interact with the web server from Javascript - there is lots of potential fun
- Different browsers did things a *little* differently - this led to the rise of Ajax Libraries

Ajax Libraries

- Prototype - Basic portability and common functionality
  - http://www.prototypejs.org/
- Script.aculo.us - Animation and effects
  - http://script.aculo.us/
- Jquery - General purpose - http://jquery.com/
- Google Web Toolkit - http://code.google.com/webtoolkit/

Updating your page dynamically with `Ajax.Updater`

Developers often want to make Ajax requests to receive HTML fragments that update parts of the document. With `Ajax.Request` with an `onComplete` callback this is fairly easy, but with `Ajax.Updater` it's even easier!

Suppose you have this code in your HTML document:

```
<h2>Our fantastic products</h2>
<div id="products">(fetching product list ...)</div>
```

Text

The 'products' container is empty and you want to fill it with HTML returned from an Ajax response. No problem:

```
new Ajax.Updater('products', '/some_url', { method: 'get' });
```

http://www.prototypejs.org/learn/introduction-to-ajax

Ajax on Rails

- Ruby on Rails uses Prototype and Script.aculo.us
- It is tightly integrated into Rails - makes it even easier
- Using Ajax in Rails is very similar to just using Rails - as much is done for you as possible

```
<% form_remote_for :umap, @user, :update=>
'main-content',  :url => { :action => "add" } do |f| %>
<p>Name:
<%= f.text_field :name %> </p>
<p>Login:
<%= f.text_field :login %> </p>
<p>Password:
<%= f.password_field :password %> </p>
<p>Email:
<%= f.text_field :email %> </p>
<%= submit_tag "Create" %>
<%= submit_tag "Cancel" %>
<% end %>
```

What is different between the normal and Ajax-enabled views?

```
<% form_for :umap, @user, :url => { :action =>
"adduser" } do |f| %>
<p>Name:
<%= f.text_field :name %> </p>
<p>Login:
<%= f.text_field :login %> </p>
<p>Password:
<%= f.password_field :password %> </p>
<p>Email:
<%= f.text_field :email %> </p>
<%= submit_tag "Create" %>
<%= submit_tag "Cancel" %>
<% end %>
```

```
<% form_remote_for :umap, @user, :update=>
'main-content',  :url => { :action => "add" } do |f| %>
<p>Name:
<%= f.text_field :name %> </p>
<p>Login:
<%= f.text_field :login %> </p>
<p>Password:
<%= f.password_field :password %> </p>
<p>Email:
<%= f.text_field :email %> </p>
<%= submit_tag "Create" %>
<%= submit_tag "Cancel" %>
<% end %>
```

```
<% form_for :umap, @user, :url => { :action =>
"adduser" } do |f| %>
<p>Name:
<%= f.text_field :name %> </p>
<p>Login:
<%= f.text_field :login %> </p>
<p>Password:
<%= f.password_field :password %> </p>
<p>Email:
<%= f.text_field :email %> </p>
<%= submit_tag "Create" %>
<%= submit_tag "Cancel" %>
<% end %>
```

```
<form action="/authn/adduser" method="post">
<p>Name:
<input id="umap_name" name="umap[name]" size="30" type="text" /> </p>
<p>Login:
<input id="umap_login" name="umap[login]" size="30" type="text" /> </p>
<p>Password:
<input id="umap_password" name="umap[password]" size="30" type="password" /> </p>
<p>Email:
<input id="umap_email" name="umap[email]" size="30" type="text" /> </p>
<input name="commit" type="submit" value="Create" />
<input name="commit" type="submit" value="Cancel" />
</form>
```

Normal Form Source Code

Text

```
<form action="/users/add" method="post" onsubmit="new Ajax.Updater('main-content', '/
users/add', {asynchronous:true, evalScripts:true, parameters:Form.serialize(this)}); return false;">
<p>Name:
<input id="umap_name" name="umap[name]" size="30" type="text" /> </p>
<p>Login:
<input id="umap_login" name="umap[login]" size="30" type="text" /> </p>
<p>Password:
<input id="umap_password" name="umap[password]" size="30" type="password" /> </p>
<p>Email:
<input id="umap_email" name="umap[email]" size="30" type="text" /> </p>
<input name="commit" type="submit" value="Create" />
<input name="commit" type="submit" value="Cancel" />
</form>
```

Form Source Code Using Ajax through Prototype

Some Rails Ajax Helpers

- form_remote_for (and form_remote_tag)
- link_to_remote

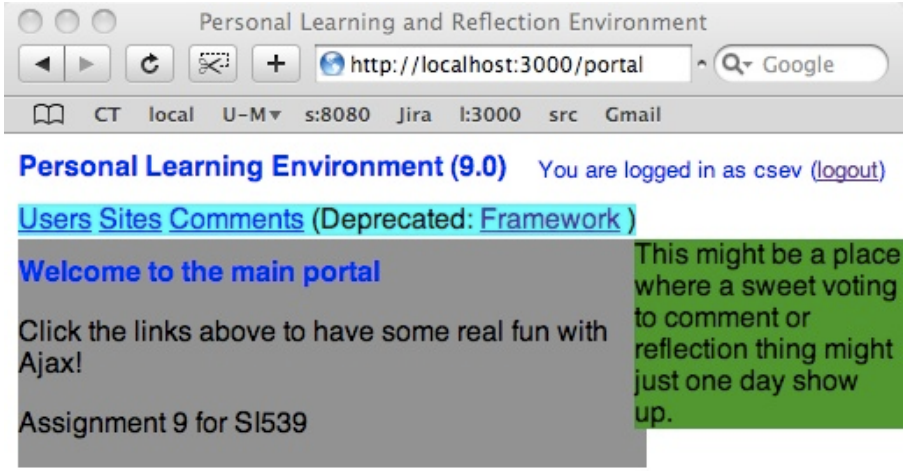
```
<%= link_to_remote 'View', :update=> 'main-content',
  :url => { :action => 'view', :id => user } %> /

<%= link_to_remote 'Delete', :update=> 'main-content',
  :url=> { :action => 'delete', :id => user },
  :confirm => 'Are you sure?', :method => :post %>

<%= link_to_remote 'Add Account', :update=> 'main-content',
   :url => { :action => 'add' } %>

<%= link_to_remote 'Sites', :update=> 'main-content',
   :url => { :action => 'ajaxstart', :controller=>'sites' } %>
```

http://api.rubyonrails.org/classes/ActionView/Helpers/PrototypeHelper.html#M000533

views/portal/index.rhtml

```
<div id="topnav">
<%= link_to_remote 'Users', :update=> 'main-content',
      :url => { :action => 'ajaxstart', :controller=>'users' } %>
<%= link_to_remote 'Sites', :update=> 'main-content',
      :url => { :action => 'ajaxstart', :controller=>'sites' } %>
<%= link_to_remote 'Comments', :update=> 'right-content',
      :url => { :action => 'ajaxstart', :controller=>'comments' } %>
</div>
<div id="main-content">
Welcome to...
</div>
```

link_to_remote outside the div



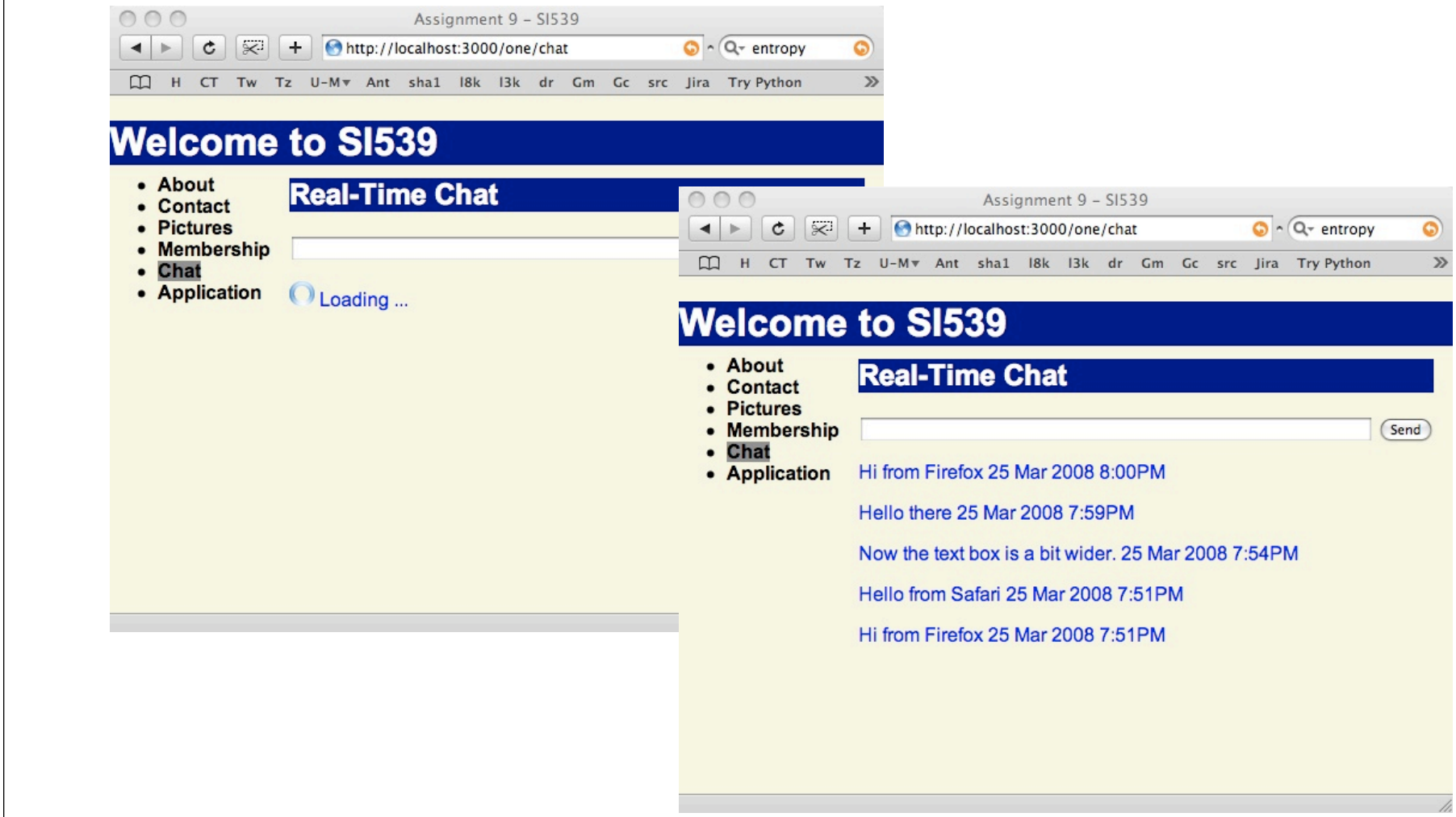
```
<% form_remote_for :umap, @user,
  :update=> 'main-content',
  :url => { :action => "add" } do |f| %>
```

http://api.rubyonrails.org/classes/ActionView/Helpers/PrototypeHelper.html#M000535

A Real-Time Chat Sample

Topics

- Periodic Updating
- Form Submission into an Ajax Form
- Rendering without layout
- created_at convention in a model


Assignment 9 – SI539
http://localhost:3000/one/chat
entropy
H CT Tw Tz U-M Ant sha1 l8k l3k dr Gm Gc src Jira Try Python
Welcome to SI539
About
Contact
Pictures
Membership
Chat
Application
Real-Time Chat
Loading ...
Assignment 9 – SI539
http://localhost:3000/one/chat
entropy
H CT Tw Tz U-M Ant sha1 l8k l3k dr Gm Gc src Jira Try Python
Welcome to SI539
About
Contact
Pictures
Membership
Chat
Application
Real-Time Chat
Send
Hi from Firefox 25 Mar 2008 8:00PM
Hello there 25 Mar 2008 7:59PM
Now the text box is a bit wider. 25 Mar 2008 7:54PM
Hello from Safari 25 Mar 2008 7:51PM
Hi from Firefox 25 Mar 2008 7:51PM

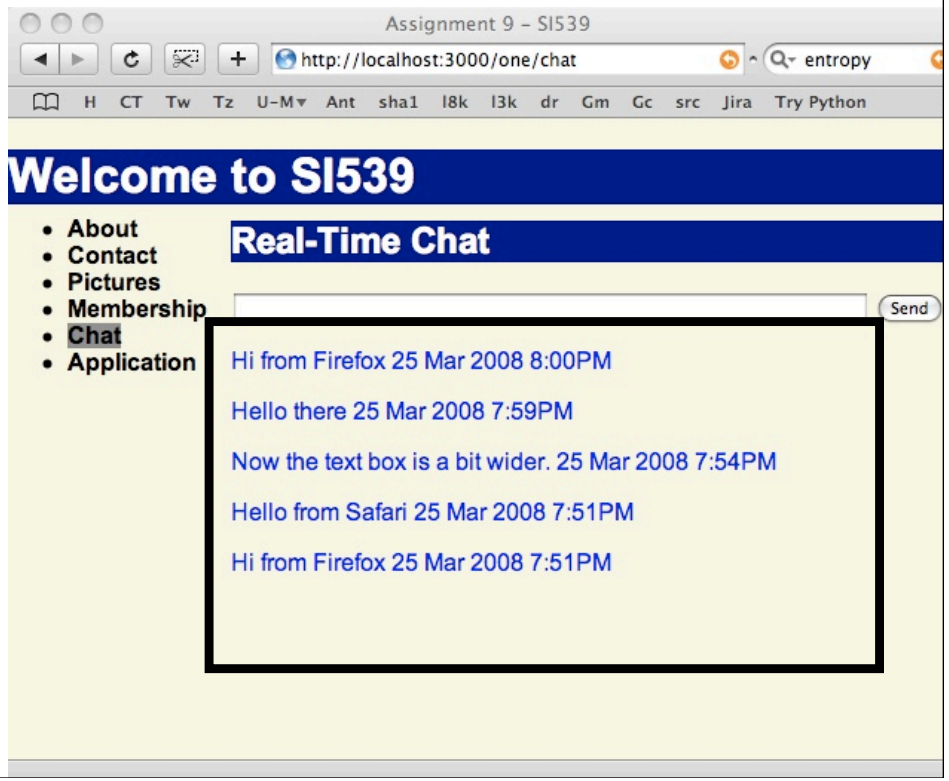
```
<h2>Real-Time Chat</h2>
<p>
<!-- http://api.rubyonrails.org/classes/ActionView/Helpers/PrototypeHelper.html -->
 <% form_remote_tag :url => 'chatcontent', :update => 'chatdiv' do -%>
   <input type="text" size="60" name="chatmsg"/>
   <%= submit_tag 'Send' %>
 <% end %>
</p>
<!-- http://api.rubyonrails.org/classes/ActionView/Helpers/PrototypeHelper.html -->
<%=  periodically_call_remote(:url => 'chatcontent',
      :frequency => '3', :update => 'chatdiv') %>
<div id="chatdiv">
<%= image_tag "ajax-loading.gif" %>
Loading ...
</div>
```

```
<h2>Real-Time Chat</h2>
<p><form action="chatcontent" method="post" onsubmit="new
Ajax.Updater('chatdiv', 'chatcontent', {asynchronous:true, evalScripts:true,
parameters:Form.serialize(this)}); return false;">    <input type="text" size="60"
name="chatmsg"/>
   <input name="commit" type="submit" value="Send" />
 </form></p>
<script type="text/javascript">
//<![CDATA[
new PeriodicalExecuter(function() {new Ajax.Updater('chatcontent', 'chatdiv',
{asynchronous:true, evalScripts:true})}, 30)
//]]>
</script>
<div id="chatdiv">
<img alt="Ajax-loading" src="/images/ajax-loading.gif?1206489556" />
Loading ...
</div>
```

```
def chatcontent
   if request.post? and params[:chatmsg] != nil
       logger.info "Chat"
      ch = Chat.new
      ch.chatmsg = params[:chatmsg]
      ch.save
   end
   @chats = Chat.find(:all, :order => "chats.created_at DESC",
        :limit => 5)
   logger.info "We found #{@chats.size} chats"
   render :action => 'chatcontent', :layout => false
end
```

http://localhost:3000/one/chatcontent

```
<p class="chattext">
<span class="chatdate">
25 Mar 2008 11:50PM</span>
</p>
<p class="chattext">Hi from Firefox
<span class="chatdate">
25 Mar 2008  8:00PM</span>
</p>
<p class="chattext">Hello there
<span class="chatdate">
25 Mar 2008  7:59PM</span>
</p>
```



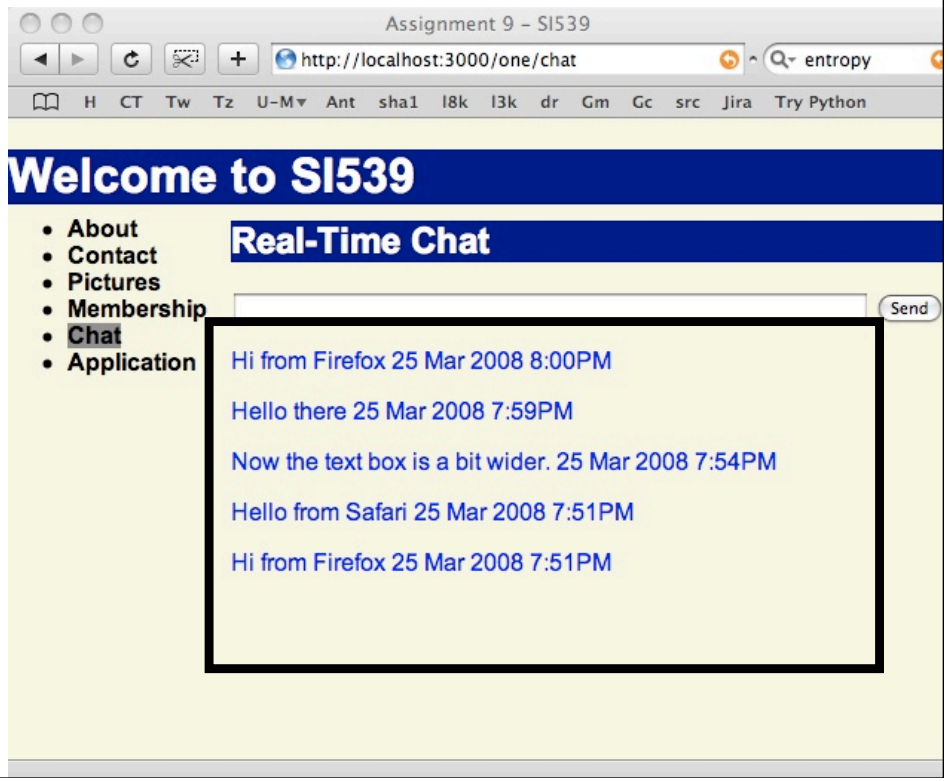
```
class CreateChats < ActiveRecord::Migration
  def self.up
    create_table :chats do |t|
      t.column :chatmsg, :string
      t.column :member_id, :integer
      t.column :created_at, :datetime
    end
  end

  def self.down
    drop_table :chats
  end
end
```

Convention - this field is handled by Rails.

```
<% for chat in @chats %>
    <p class="chattext"><%= chat.chatmsg %>
    <span class="chatdate">
        <%= chat.created_at.strftime("%e %b %Y %l:%M%p") %>
    </span>
    </p>
<% end %>
```

http://localhost:3000/one/chatcontent

```
<p class="chattext">
<span class="chatdate">
25 Mar 2008 11:50PM</span>
</p>
<p class="chattext">Hi from Firefox
<span class="chatdate">
25 Mar 2008  8:00PM</span>
</p>
<p class="chattext">Hello there
<span class="chatdate">
25 Mar 2008  7:59PM</span>
</p>
```

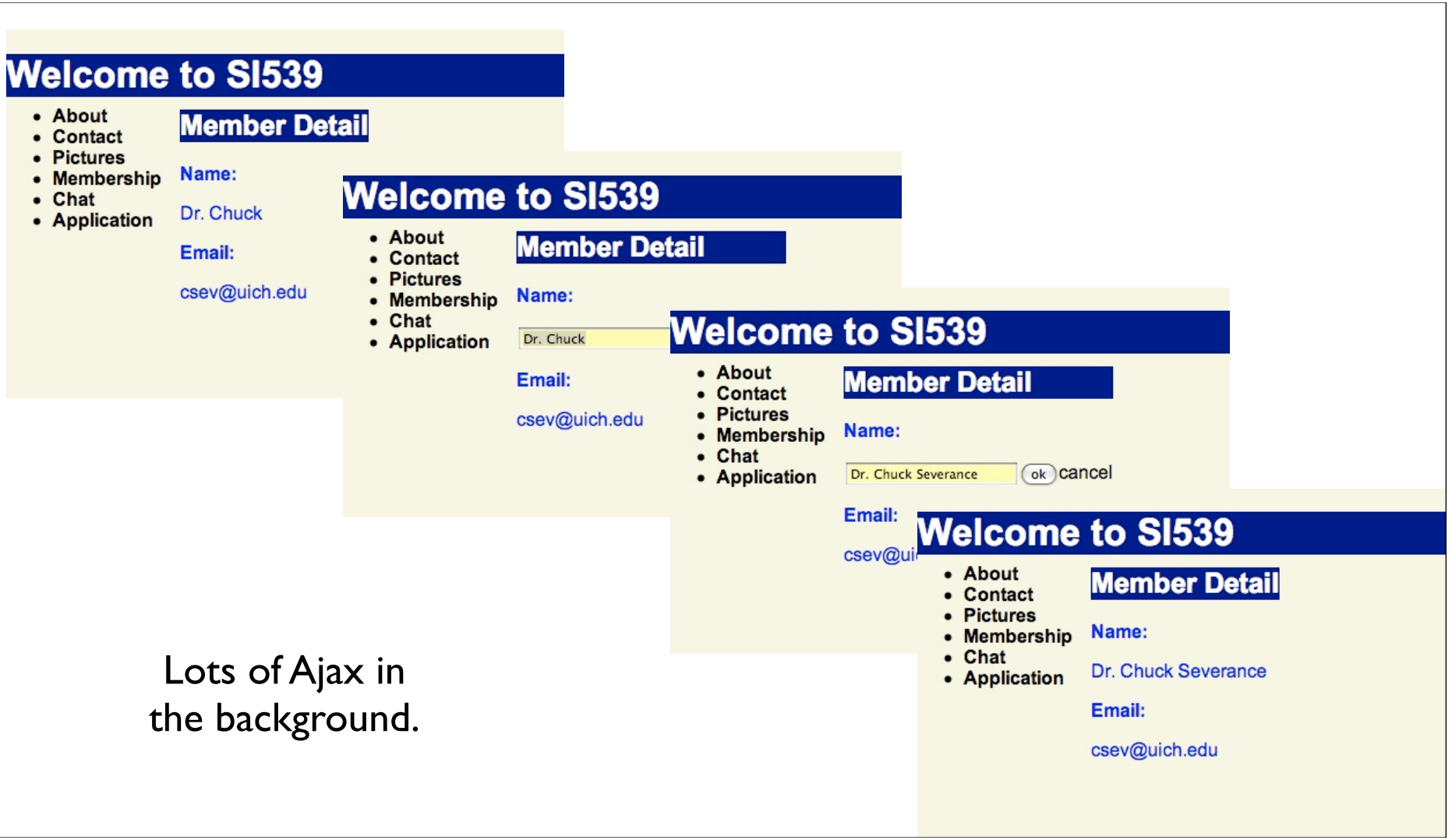


```
<h2>Real-Time Chat</h2>
<p>
<!-- http://api.rubyonrails.org/classes/ActionView/Helpers/PrototypeHelper.html -->
 <% form_remote_tag :url => 'chatcontent', :update => 'chatdiv' do -%>
   <input type="text" size="60" name="chatmsg"/>
   <%= submit_tag 'Send' %>
 <% end %>
</p>
<!-- http://api.rubyonrails.org/classes/ActionView/Helpers/PrototypeHelper.html -->
<%=  periodically_call_remote(:url => 'chatcontent',
      :frequency => '3', :update => 'chatdiv') %>
<div id="chatdiv">
<%= image_tag "ajax-loading.gif" %>
Loading ...
</div>
```

In Place Editing

In Place Editing

- This is nifty - you can do editing live - as the fields are changed the database is updated.



Lots of Ajax in the background.

```
<%= in_place_editor_field "member", "name", {}, {
  :load_text_url => url_for(:action => "get_member_name", :id =>@member)
 } %>
```

```
def view
  @member = Member.find(params[:id])
end
```

```
<%= in_place_editor_field "member", "name", {}, {
  :load_text_url => url_for(:action => "get_member_name", :id =>@member)
 } %>
```

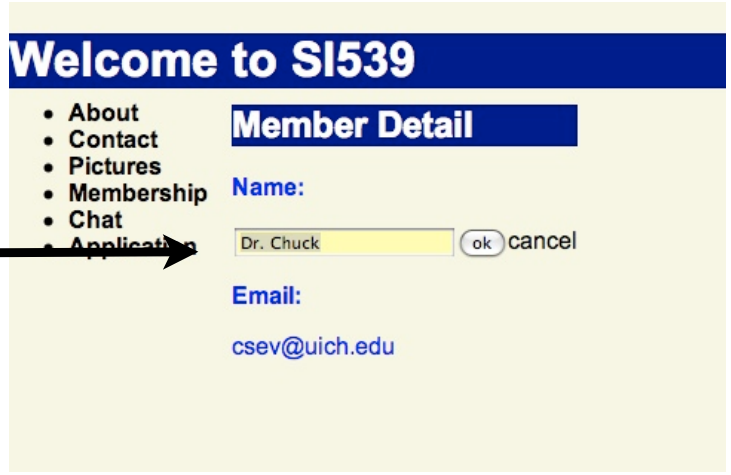
```
<span class="in_place_editor_field" id="member_name_1_in_place_editor">
Chuck S</span><script type="text/javascript">
//<![CDATA[
new Ajax.InPlaceEditor('member_name_1_in_place_editor',
'/one/set_member_name/1', {loadTextURL:'/one/get_member_name/1'})
//]]>
</script>
</span>
```

views/one/view.rhtml

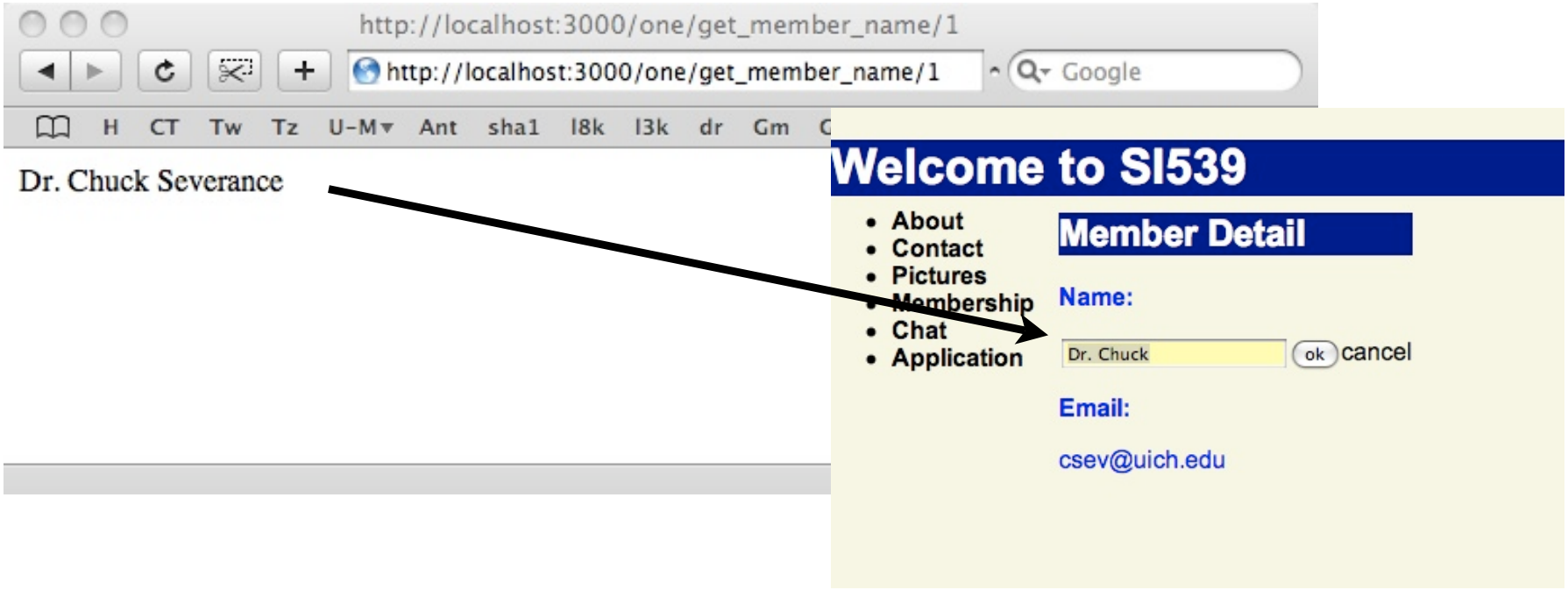
```
<%= in_place_editor_field "member", "name", {}, {
  :load_text_url => url_for(:action => "get_member_name", :id =>@member)
 } %>
```

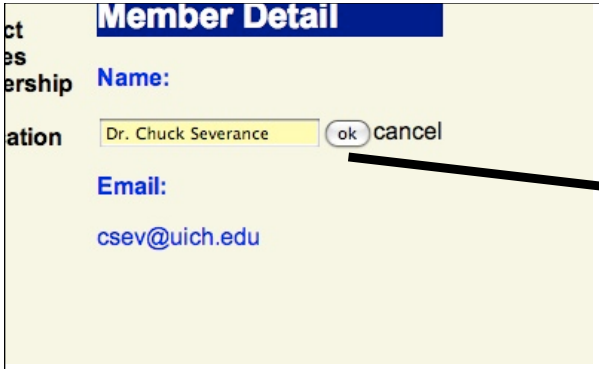
```
def get_member_name
  item = Member.find(params[:id])
  render :text => item.name.to_s
end
```

one_controller.rb



```
def get_member_name
  item = Member.find(params[:id])
  render :text => item.name.to_s
end
```





one_controller.rb

```
def set_member_name
  item = Member.find(params[:id])
  item.update_attribute(:email, params[:value])
  render :text => item.email.to_s
end
```

The moment you press "OK", a post is sent to the set_member_name method with the members's id and the new field value from the form. The database is immediately updated and the new text is returned to the form.

user_controller.rb

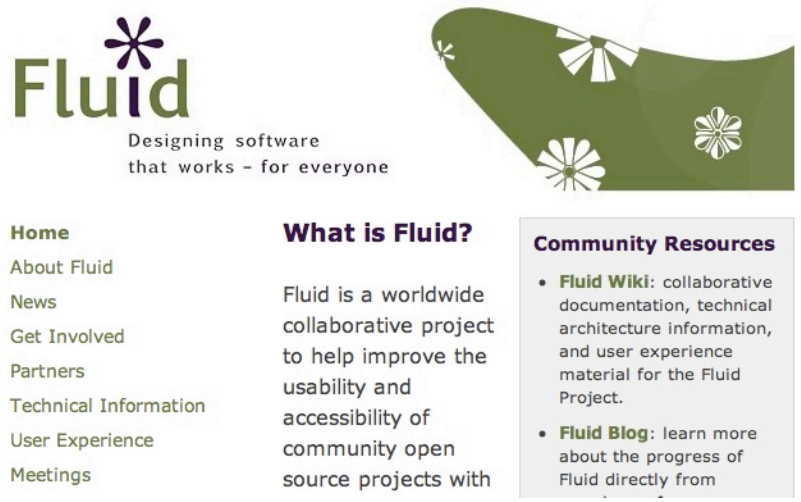
```
in_place_edit_for :member, :name

def set_member_email
  logger.info "Doing set_member_email manually"
  item = Member.find(params[:id])
  item.update_attribute(:email, params[:value])
  render :text => item.email.to_s
end

def get_member_name
  item = Member.find(params[:id])
  render :text => item.name.to_s
end

def get_member_email
  item = Member.find(params[:id])
  render :text => item.email.to_s
end
```

No shortcut for getters. Hmmm.

Kind of WET (not-DRY)

Accessibility with Ajax

- It is not perfect
- Needs further study
- Lots of investment going on
- Fluid Project - Univ. Toronto
  - http://www.fluidproject.org/



Summary

- By using the Protoype library and closely integrating it into Rails, Ajax becomes very simple in Rails.
- We may have a whole new set of Ajax-enabled applications emerge as a result
- This will expose new design issues - users will see and learn new ways of interacting - there will be usability advantages and disadvantages.

Assignment 9

- Ajax real time chat
- Ajax editing on the view
- Unit tests